RED HAT®
ENTERPRISE LINUX®
DEVELOPERS GROUP

Eclipse for C/C++ Developers

Using Red Hat Enterprise Linux

Developer Tools

Red Hat Developer Day

Jeff Johnston

26 June 2012

redhat.

- What is Eclipse?
  - Often thought of as an IDE
  - Actually a framework
  - Supports plug-in functionality
  - Plug-ins are grouped into features
  - Features/plug-ins may have other feature/plug-in dependencies
  - Platform is eclipse-platform package on RHEL

RED HAT®
ENTERPRISE LINUX®
DEVELOPERS PROGRAM

SEE US AT SUMMIT

Visit us at Developer Zone!

FOLLOW US ON TWITTER

twitter.com/#!/RHELdevelop

PLAY US ON YOUTUBE

bit.ly/RHELdevOnYouTube

LEARN & SHARE AT

red.ht/rheldevoug

GIVE US FEEDBACK

RHELdevelop@redhat.com

redhat.

Eclipse IDE Terminology

- Eclipse Workspace
  - Directory where Eclipse session is based
  - Workspace has “preferences”
  - Can switch between Workspaces
- Eclipse Project
  - User code is put into a Project
  - Project resides physically or virtually in Workspace
  - Project has nature(s) and builder(s)
    - Stored in special .project file
  - Project has “properties”
  - Properties can override Workspace preferences

Eclipse Terminology Continued

- Eclipse View
  - Simply a window
- Eclipse Perspective
  - Group of Views possibly with default initial layout
  - Perspective will determine convenient menu items
- Team Provider
  - Terminology used by Eclipse for Version Control System
  - Eclipse can figure this out for an existing CVS check-out

CDT (C/C++ Development Tooling)

- Set of features/plug-ins to supply C/C++ IDE
- Shipped as eclipse-cdt package in RHEL
- Edit/compile/debug/run
- C and C++ project natures
- Special .cproject file added to project
  - Contains build settings mostly

JDT (Java Development Tools)

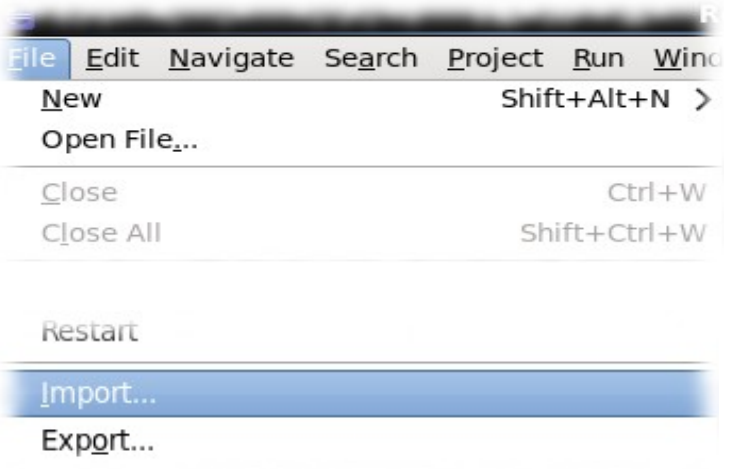
- Set of features/plug-ins to support Java IDE
- Shipped as eclipse-jdt package in RHEL
- Edit/compile/debug/run/test
- Java project nature

PDE (Plug-in Development Environment)

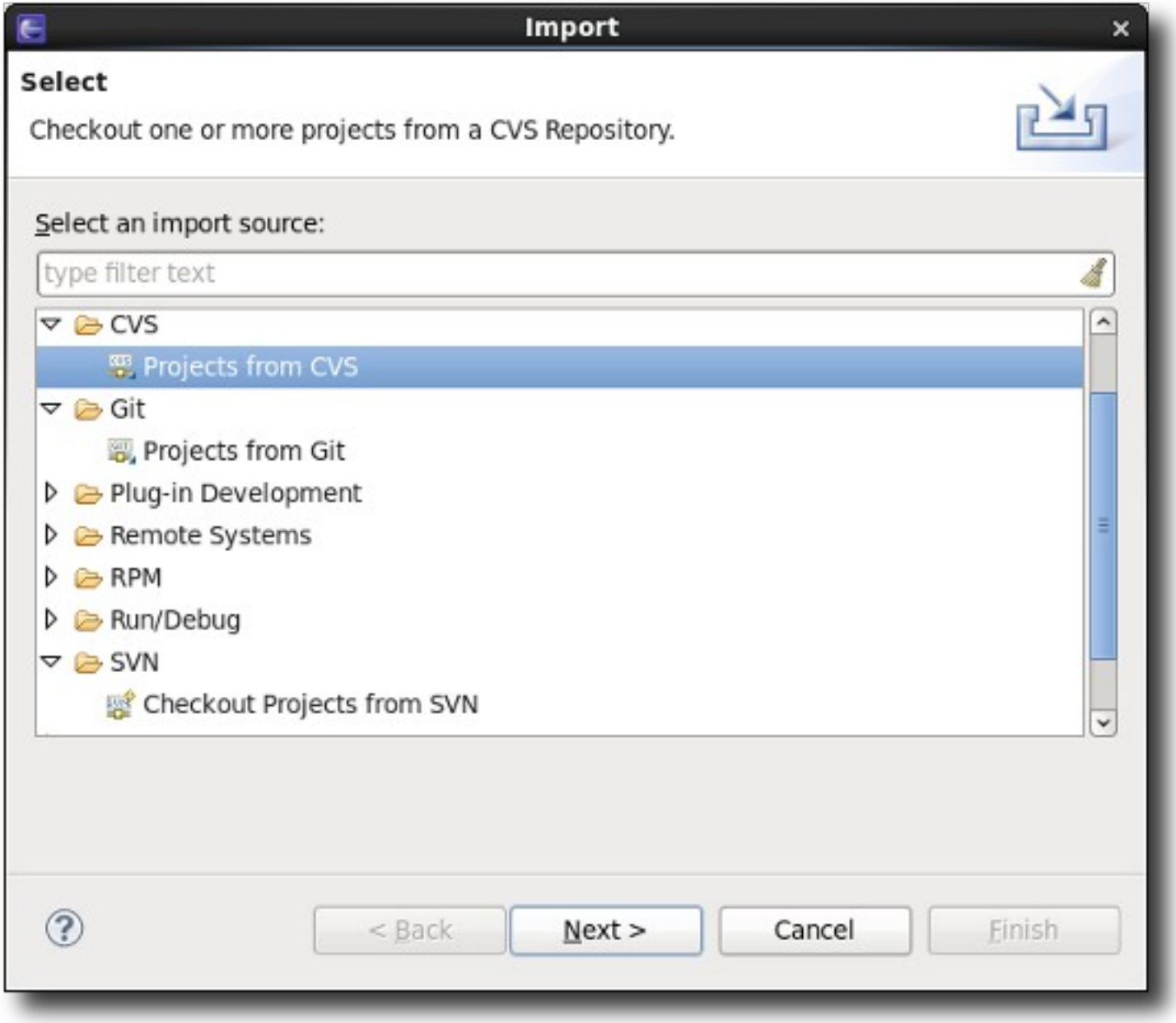
- Set of features/plug-ins to develop features/plug-ins
- Shipped as eclipse-pde package in RHEL
- Edit/compile/debug/test
- Can kick off child Eclipse session to test code
  - New session can optionally use features/plug-ins in workspace
  - Can include features/plug-ins from parent Eclipse session

Importing Code Into Eclipse

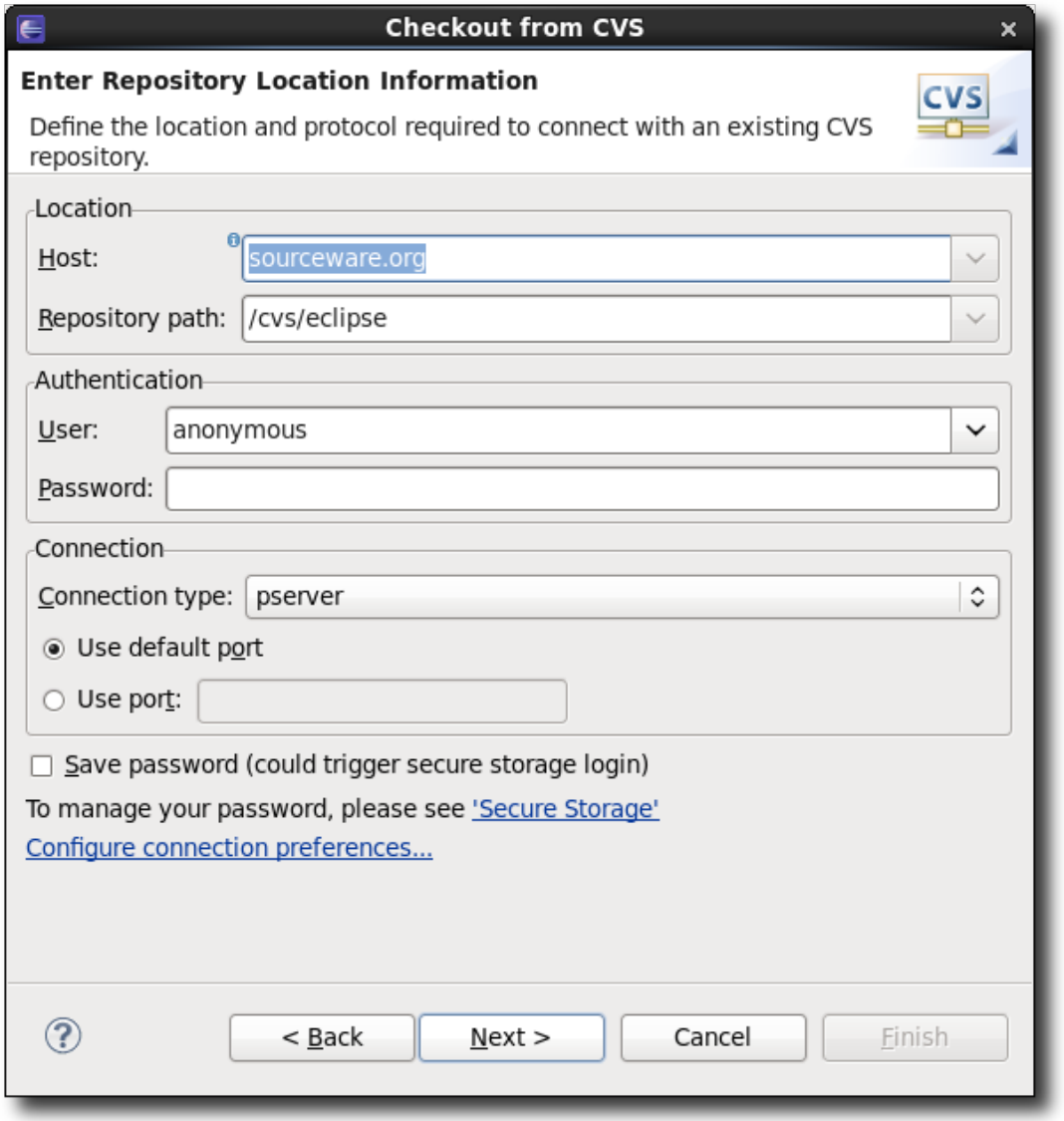
- Eclipse has support for multiple Team Providers
  - CVS is installed with main eclipse-platform package
  - SVN is available via eclipse-subclipse package in RHEL
  - Upstream support only for git
- Use import to check code out from repositories



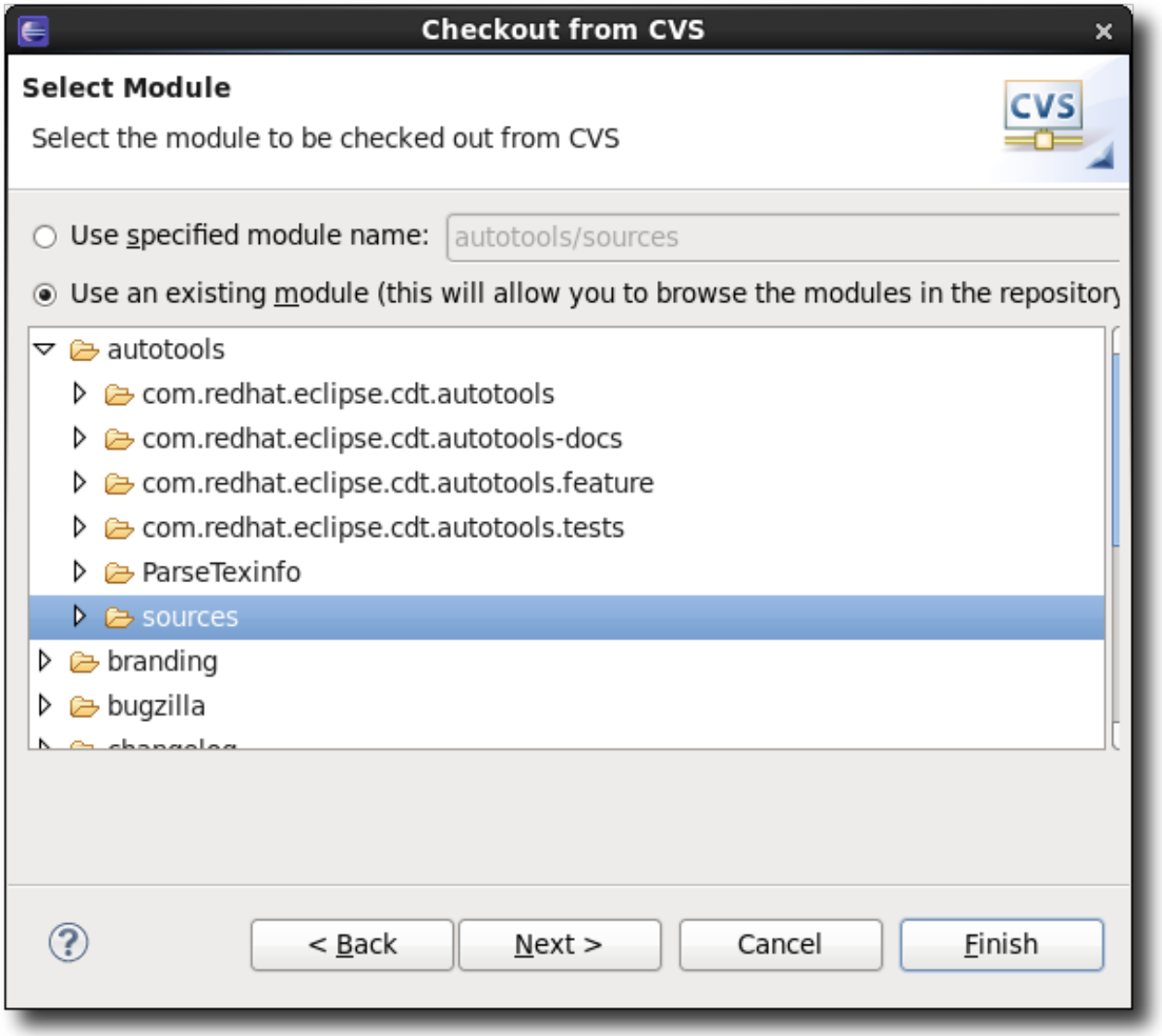
Importing Code Into Eclipse Continued..



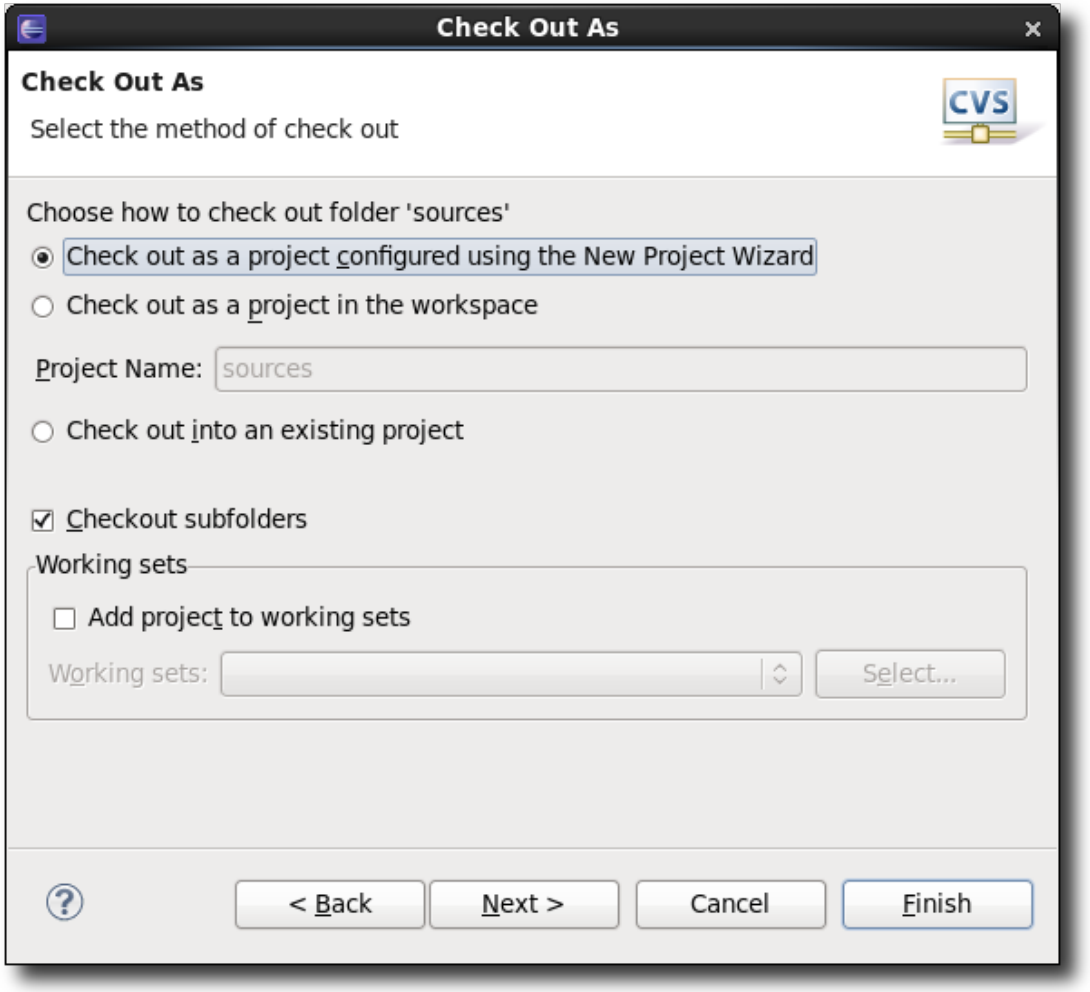
Checkout From CVS



Checkout From CVS Continued..



Check Out As - Dialog

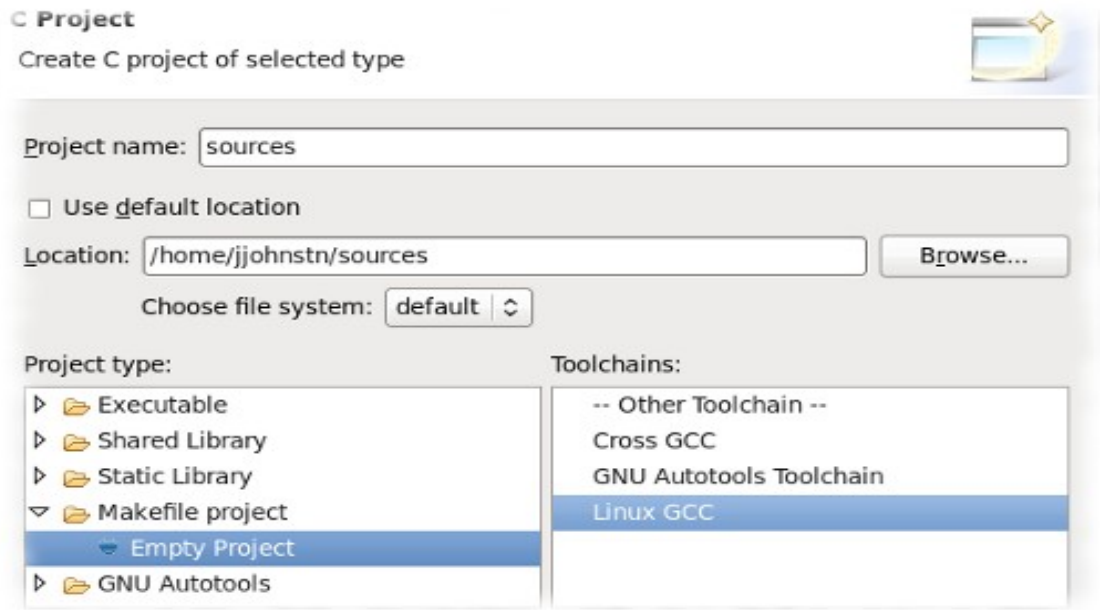


Check Out As - Dialog Continued...

- Get to this dialog by clicking Next button
- Check out as a project in workspace
  - Do this if code has checked in .project file
- Check out into an existing project
  - Do this if you have an empty project
  - Or if you want to add code to an existing project
- Check out as a project using New Project Wizard
  - Do this most of the time
  - Allows you to choose the project type
    - e.g. Choose C project if checking out C code

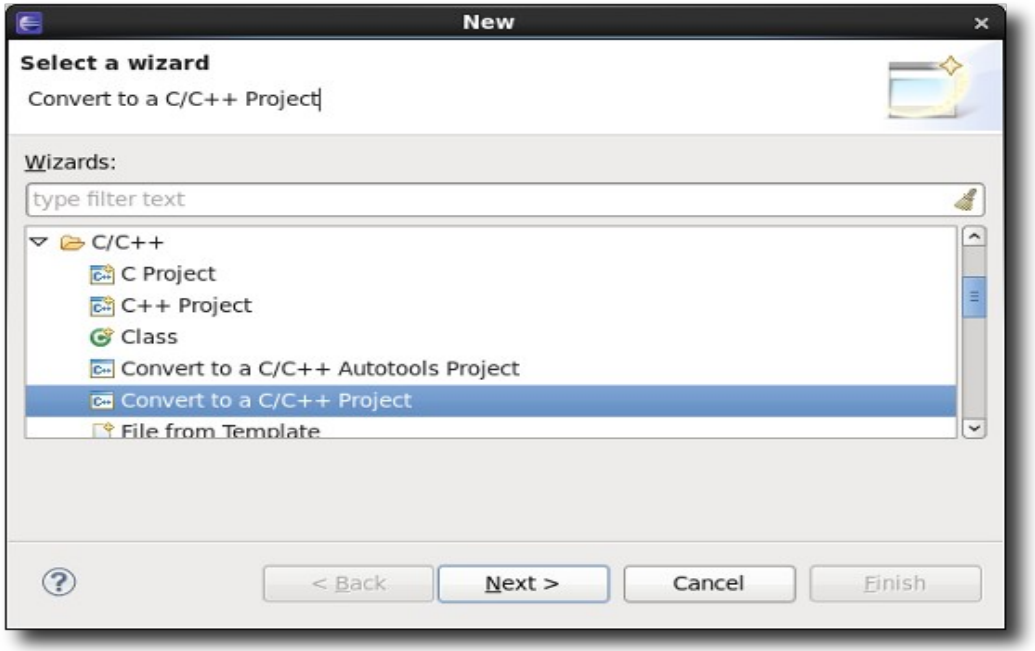
Import Alternatives

- Point to an existing check-out location
  - Eclipse projects can specify a location outside of workspace
    - Un-check “Use default location” box
    - Specify empty project template if offered



Import Alternatives Continued..

- Simply import code without using Project Wizard
  - Convert project afterwards to project type using New Wizard
  - Click File-> New -> Other... to find list of Wizards
    - e.g. C/C++ tree item contains C/C++ conversion wizards



Import Alternatives Continued...

- Create an empty project first
  - Use Team Provider to check-out into project
    - In Checkout As Dialog specify “Check out into existing project”
  - If you have a source tarball of some kind
    - File -> Import... -> General -> Archive File
    - Specify your empty project location as folder to use
- Import an existing project directly
  - Use File -> Import... -> General -> Existing Projects into Workspace
    - You can choose to copy the project or use the location
    - You can also specify an archive file
      - Can be created via File -> Export... -> General -> Archive File

C/C++ Project Types

- Three major categories
  - Managed Make Project
    - Eclipse manages your Makefile automatically
    - Project has overall target type (e.g. Executable) and tool set
    - Each tool has input types and output(s)
    - Eclipse figures out how to get from sources to target using tools
  - Standard Make Project
    - You edit and maintain your own Makefile
  - Autotools Project
    - Configuration script creates Makefile
    - You maintain files that are used to create configuration script
    - Build has additional configuration step prior to running make

Managed Make Project

- Useful for starting a project from scratch quickly
- Can opt to output Makefile
  - If you check it in, then external users can build via the Makefile
  - Otherwise, they must import the project into Eclipse and build there
- Toolset
  - Toolset is group of tools used in the Makefile
  - Each tool has options to use, accepted inputs, and outputs
    - e.g. Gcc compiler operates on .c .cpp files and outputs .o files
- Can build via make command or use internal builder

Managed Make Project Continued..,

- Can switch final target type
  - e.g. Can switch from creating an executable to a shared library
- Can get automatic settings for referenced projects
  - Referencing a Managed Make library project automatically adds the library location and header file paths to the referencing Managed Make project
- Can fine tune what directories are used for sources

Standard Make Project

- Use this for an existing run-of-the-mill Makefile project
- Tool settings aren't used or available
- You set up default build and clean make targets
  - Usually “all” and “clean”
- Can set up custom make targets
  - Right-click on project, select Make Targets -> Create...
  - Use Make Targets -> Build... to make these custom targets

Autotools Project

- Use this for projects using Autotools
  - e.g. GNU/FSF packages such as gcc, gdb, binutils, gimp, etc...
- Special Autotools options in Project properties
  - Can specify options to configure and autogen.sh scripts here
  - Also can specify location of autotools such autoconf, automake, etc..
- Special editor for editing configure.in/configure.ac
  - Hover help and completion for autoconf macros
- Top-level Make targets are scraped automatically
  - No need to create custom targets for the many generated targets

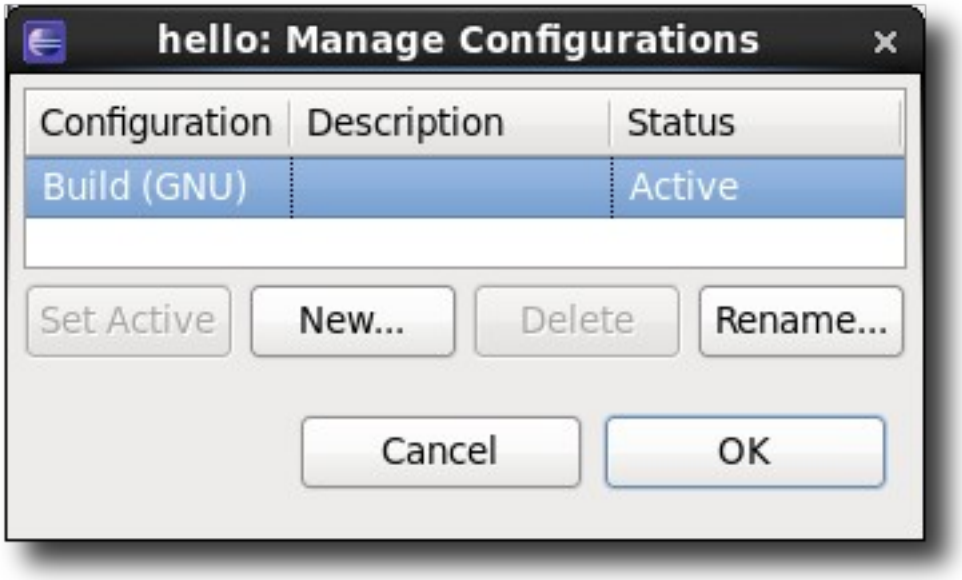
Autotools Project Continued..

- Special build step occurs before invoking make
  - Looks for configure script or autogen.sh script or it creates configure script via “autoreconf -i”
  - Future builds will verify if configure script needs to be regenerated or whether current Makefile is valid
- Autotool configuration options are per build configuration
  - Can have multiple build configurations for a project
  - Each build configuration can have separate options for running the configuration scripts
  - Useful for multiple cross-platform targets or setting up debug-enabled builds or special options

Build Configurations

- Set of options, settings pertaining to the build
  - Each configuration will have separate build directory
- Options that are per-configuration will have the configuration pull-down at the top of their settings page
- All C/C++ projects will have some default configuration
- Multiple canned configurations may be offered
  - e.g. Managed projects offer Debug and Release configurations for building with and without debug info
- User may create/rename/copy/delete configurations via the Manage Configurations dialog
- Active configuration is the one used for building

Build Configurations Continued..



Indexer

- CDT has its own indexer
- Indexer allows code traversal and searching
  - F3 - go to definition/declaration
    - If definition not known, will just go to declaration
- Refactoring needs indexer
  - Rename function/method
  - Create getter/setter etc..
- Indexer settings in Preferences/Project Properties
- Indexer needs to know include paths and symbols
  - Can be manually specified or automatically discovered

Indexer Continued..

- Indexer used in conjunction with static code analysis
  - Warnings/errors will be marked in the C/C++ editor
- Can manually request indexer refresh
  - Right-click on any object in project and select Index menu item
  - Choice of Update with Modified Files or Freshen All Files

Scanner Discovery

- Special builder that discovers include paths and symbols for indexer to use
- By default, discovery is automated and will parse build output to find compiler options that pertain to header paths and defined symbols
- Default include path and defined symbols for compiler found by invoking the compiler with special options
  - e.g. gcc -E -P -v -dD
- Options found under Project Properties
  - C/C++ Build -> Discovery Options
  - Recommended to leave default options in place

Scanner Discovery Continued..

- Automated discovery requires successful build
  - Also requires the output of the build to get compile options
    - Don't suppress build output (e.g. .silent)
- Paths and symbols are per C/C++ file being compiled
  - Header files indexed on first usage by a compile unit
- Default paths and symbols found in Project Properties
  - C/C++ General -> Paths and Symbols
  - Includes tab contains default compiler header paths used
  - Symbols tab contains internal symbols generated by compiler
  - Both can be edited manually to add additional entries

Build Variables

- Macros defined to be used during the build
- Can be specified in build options
  - Can also be used to form other build variables
- Default build macros are provided (system variables)
  - Workspace location, project location, project name
  - Active configuration
- Not exposed to external environment
  - Use Environment variables instead for that purpose
- Found under Project Properties
  - C/C++ Build -> Build Variables

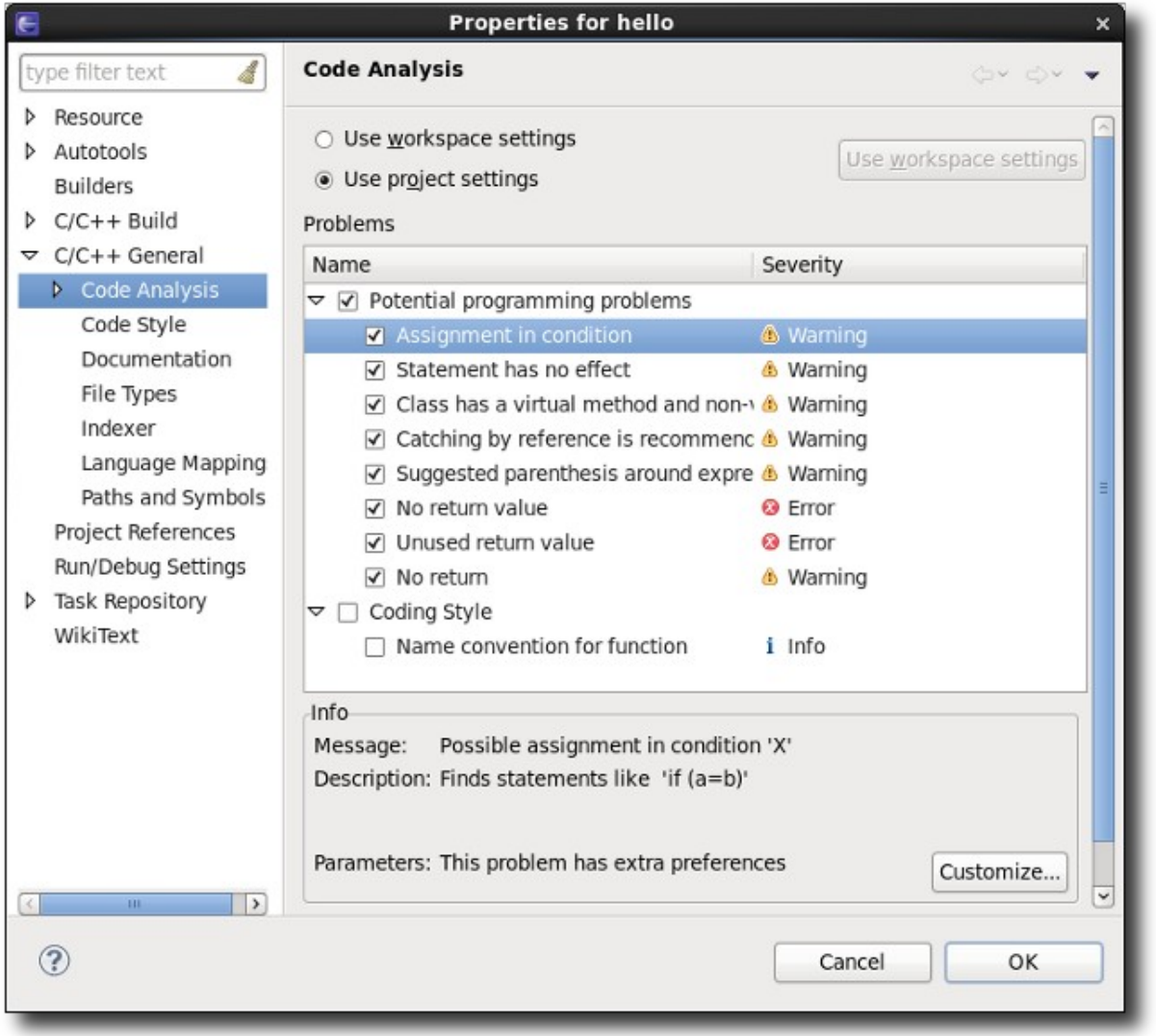
Environment

- User can set environment variables during build
  - Variables are used to set environment when running external commands
  - e.g. Running make or gcc or configure
- Environment set is not used for launching executables
  - Need to set special environment for launching
- Set via Project Properties
  - C/C++ Build -> Environment
  - Can append or replace current environment

Code Analysis

- Static analysis
- Configurable under Project -> Properties
  - C/C++ General -> Code Analysis
  - Number of default tests provided (e.g. Look for if (a = b))
  - Can configure severity or if they should be ignored
  - Tests can be added
    - Requires Java programming
- Special markers in C/C++ editor

Code Analysis Settings



C/C++ Editor

- Colourization and outline view provided
- Displays error/warning markers
  - Markers appear in margins
  - Code highlighted as well
- Ctrl + space completion
  - With libhover feature installed, includes glibc library functions
  - Will show C/C++ constructs from code and header files
  - Can also supply code templates (e.g. A for loop or case statement)
- Hover help provided
  - Same as with Ctrl + space

C/C++ Editor Continued..

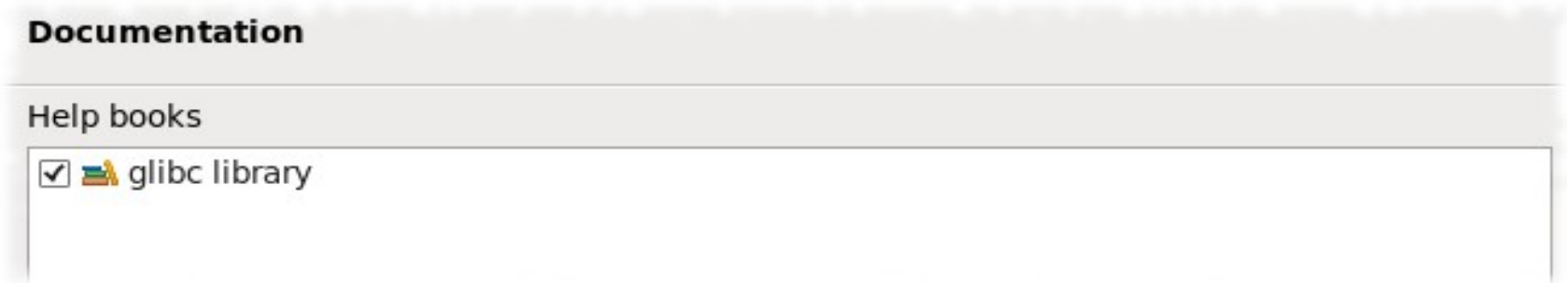
- Ctrl + Shift + N (add include)
  - For glibc C functions, highlighting and using CTRL+Shift+N will add the appropriate header file
- Ctrl + Shift + F (format code)
  - Preferences/Project Properties (C/C++ General -> Code Style)
  - Can choose K&R / GNU / Whitesmiths / BSD
- Ctrl + / (toggle comment)
- Ctrl + I (correct line indentation)
- Shift + Alt + R (Rename...)
  - Rename variables, functions, methods, macros

C/C++ Editor Settings

- Window -> Preferences -> C/C++ -> Editor
- Content Assist (can control what is offered)
  - e.g. Can disable code template proposals for Ctrl + space
- Folding
  - Control what C/C++ constructs can be condensed/folded in editor
- Hovers
  - Control what hovers are offered (default is combined hover)
- Templates
  - List of code templates (e.g. A for loop, case statement)
  - Can create custom templates or disable ones already provided

Library Hover

- Libhover feature
  - Adds hover help for a library to the editor
  - Includes code completion and adding header file (C only)
- Currently only glibc library provided
- Can enable/disable individual library hovers
  - Project -> Properties -> C/C++ General -> Documentation
  - Each library hover will appear in Help books list
    - Use check-box to enable/disable

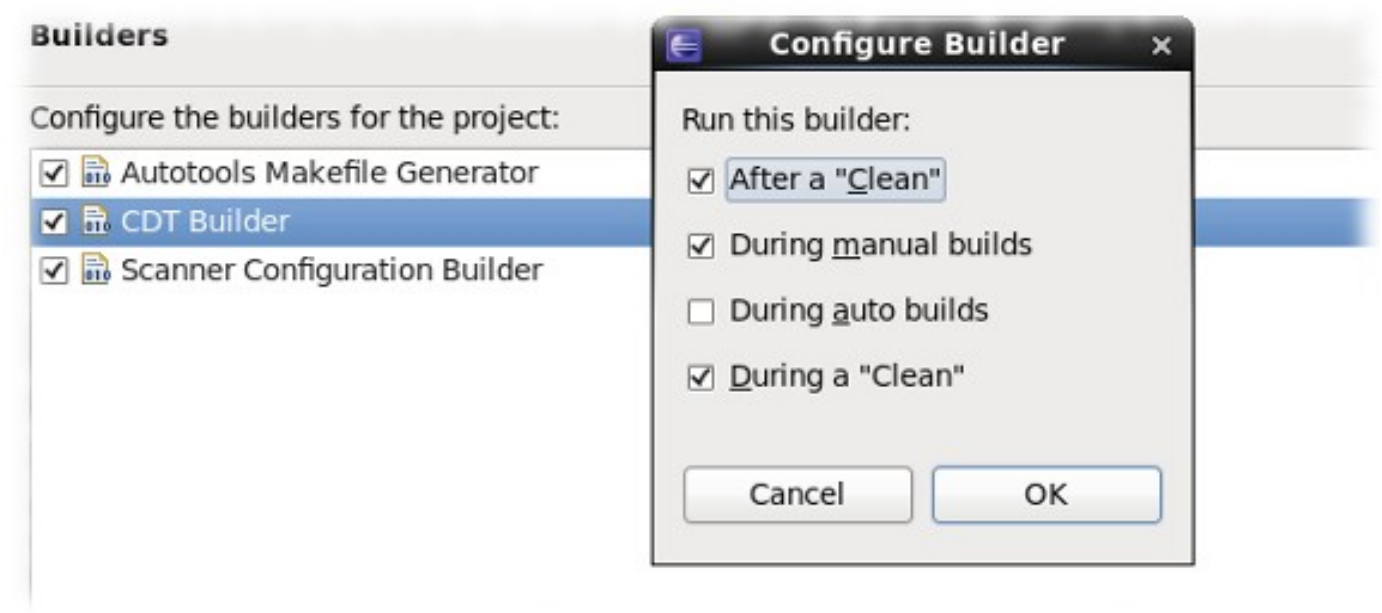


Outline View

- Companion to editor view
- Condenses into constructs
  - e.g. C/C++ editor shows functions, members, structs, etc...
- Can traverse in editor by clicking on outline item
  - Double-clicking a header file opens it in the editor
- Can sort in alphabetical order
  - Useful for searching for methods in a class
- Ctrl + O in editor gives quick outline view
  - Useful for screen real estate...close outline view and use as needed

Building

- Build uses active configuration
- Automatic builds occur when modified resource saved
  - Project -> Build Automatically flag turned on by default
  - Automatic builds do not build C/C++ projects by default
    - Project -> Properties -> Builders (select CDT Builder and Edit...)

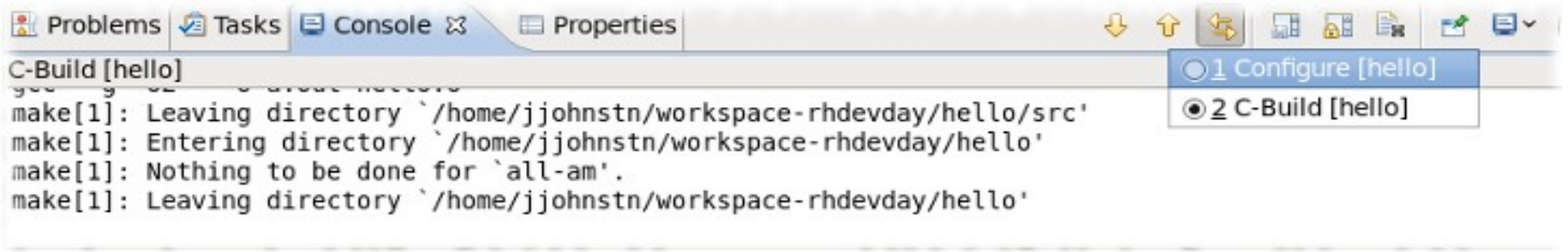


Building Continued..

- Starting a manual build
  - Project -> Build Project
  - or... clicking the Hammer icon
  - or... using a Make Target (Autotools and Standard Makefile projects)
    - Right-click Make Targets -> Build...
    - or... Project -> Make Target -> Build...
- Can set up build on resource save
  - Project Properties -> C/C++ Build
    - Behaviour tab has “Build on resource save (Auto build)” check-box
      - Not particularly recommended, default is off

Build Console

- Results of build appear in Console tab
- C/C++ builds (make) go into C-Build console
- Autotool configuration goes into Configure console
- Autotool invocation goes in Autotools console
- Each console is per project (e.g. C-Build [hello])
- Console tab has pull-down (console icon)



Error parsers

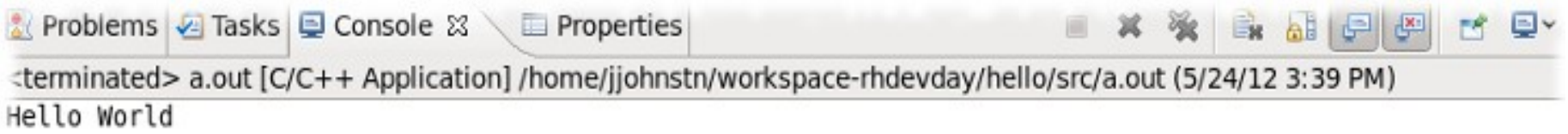
- Build output is parsed via various error parsers
- Parsers look for errors in console output
  - There is a GNU Make error parser that finds make errors
  - There is a GNU C/C++ error parser that finds GNU compiler errors
- Errors found in parsers result in markers in editor
- Errors are also posted to Problems tab
  - Double-clicking on problem in tab will go to line and file if recorded
- Error parsers can be enabled/disabled/reordered
  - Project -> Properties -> Settings -> Error Parsers tab

Binary Parsers

- Output binaries are parsed as well
- Binary parsers can be enabled/disabled/reordered
  - Project -> Properties -> Settings -> Binary Parsers tab
- Binary parsing allows one to open up a binary in Project Explorer and see headers, source files, etc..
- Binaries for last build can be found in Binaries folder
  - Folder is virtual container (i.e. No directory in workspace or project)
  - Can track down real binary in build configuration directory
    - Clicking on binary in Binaries folder will show project path on bottom of Eclipse window

Running your executable

- Right-click on executable in Project Explorer
  - Select Run as -> Local C/C++ Application
  - First time will ask if you want to use gdb/mi or gdbserver or remote gdb/mi
    - In most cases just hit enter or click on OK and select gdb/mi
  - Using Run icon runs last executable or use pull-down arrow to select
  - Stdin will fetch input from Console tab
  - Stdout/stderr output will be presented in Console tab



Running Your Executable Continued...

- For more complex cases
  - Right-click on executable and select Run as.. -> Run Configurations...
  - This brings up the Run Configurations dialog
  - Can select type of executable
    - Should default to C/C++ Application
  - Can add arguments for main() via Arguments tab
  - Can specify environment variables via Environment tab
    - Note that build environment variables are not the same
    - Cannot specify build variables for environment variable values
    - Set of default variables you can work with (e.g. workspace_loc)
    - Can append to native environment or replace it
    - specify LD_LIBARY_PATH for libraries in your workspace

Debugging Your Executable

- Right-click on executable in Project Explorer
  - Select Debug as -> Local C/C++ Application
  - Will be prompted to bring up Debug Perspective
  - Using Debug icon debugs last executable or use pull-down arrow
- Set break-points by double-clicking editor left margin
  - Breakpoints are marked in C/C++ editor with blue dot in margin
- Right-click in left margin allows more options
  - Toggle enablement / delete the breakpoint
  - Breakpoint Properties... allows further details
    - Condition: set a boolean condition to enable the breakpoint
    - Ignore count: set a number of times to continue before stopping
    - Can also set actions for when the breakpoint is triggered

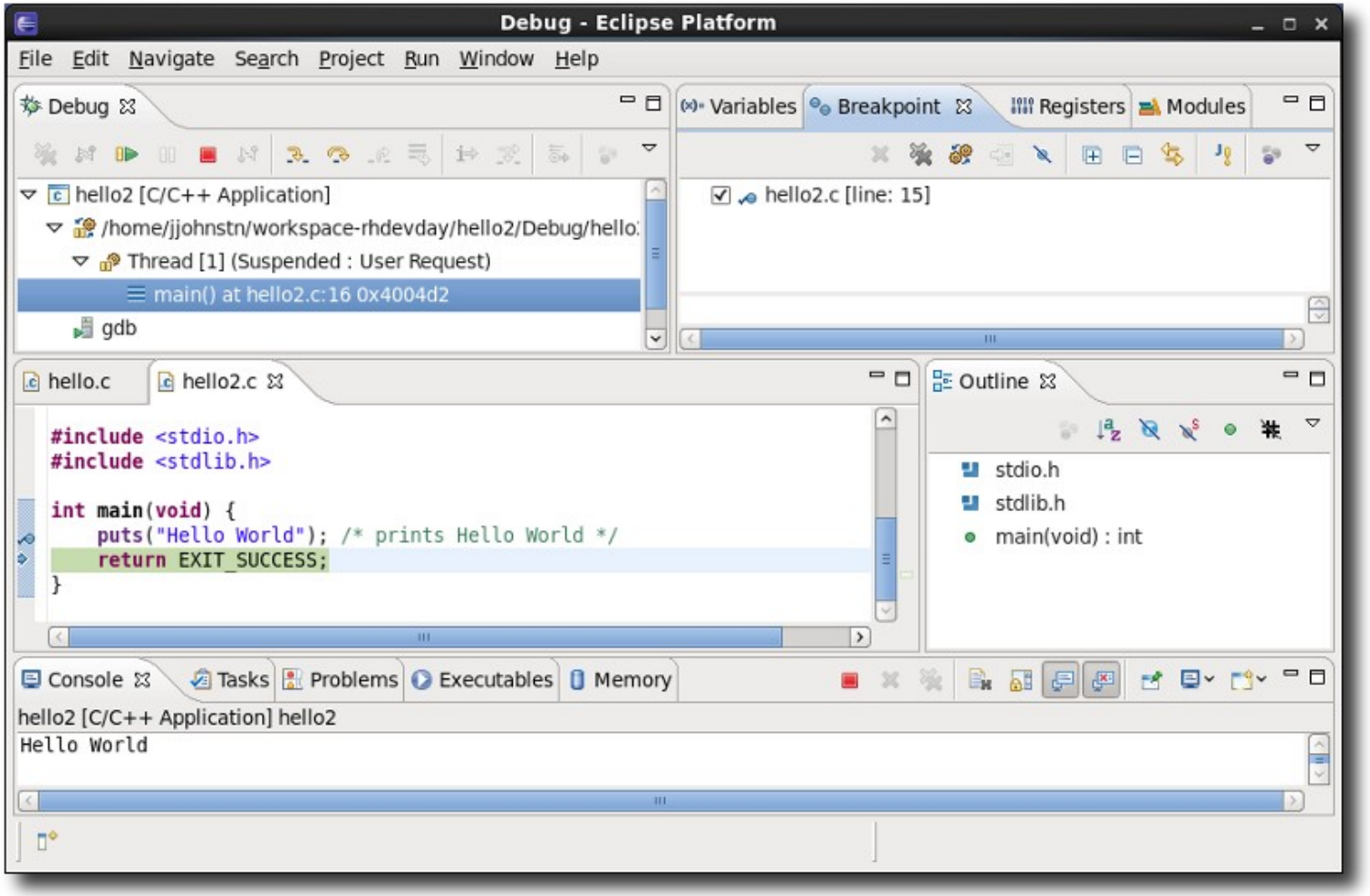
Debugging Your Executable Continued..

- For more complex cases
  - Right-click executable and select Debug as.. -> Debug Configurations...
  - Debug configurations and run configurations are tied together
    - Argument to main, environment variables, ... are the same
  - Can attach to an executing application (e.g. Debug Firefox)
  - Can specify a remote application
  - Can look at a core dump
  - Debugger tab allows debugger settings
    - Non-stop mode for thread debugging, reverse debugging
    - Shared libraries
    - can specify location of gdb and .gdbinit file
      - For special cases specify a macro for gdb which calls the debugger e.g. Use this to log gdb/mi output for a bug report

Debug Perspective

- Editor showing current line and breakpoints
- Debug View controls debug flow
  - Step Over Step Into Step Return Resume Stop
  - Shows your application plus trace-back including threads
  - Shows the gdb executable
    - Clicking on gdb executable allows you to enter and see output of gdb commands in the Console view (e.g. You can print x)
- Variables tab shows application variables and values
- Breakpoint tab shows list of breakpoints and status
- Registers tab shows system registers
- Modules tab shows dynamic libraries loaded

Debug Perspective Continued..



Profiling Your Executable

- Right-click on executable and select Profile As...
  - Profile button profiles last executable or use pull-down arrow
- Multiple profile tools supported
  - Valgrind (memory allocation profiling)
    - Provided in RHEL eclipse-valgrind package
  - Oprofile (requires hardware counters so won't work under VM)
    - Provided in RHEL eclipse-oprofile package
  - Oprofile (manual) – use this if you want to control the daemon manually
  - Function Callgraph (function call tracing)
    - Provided in RHEL eclipse-callgraph package

Valgrind (Memory Allocation Profiling)

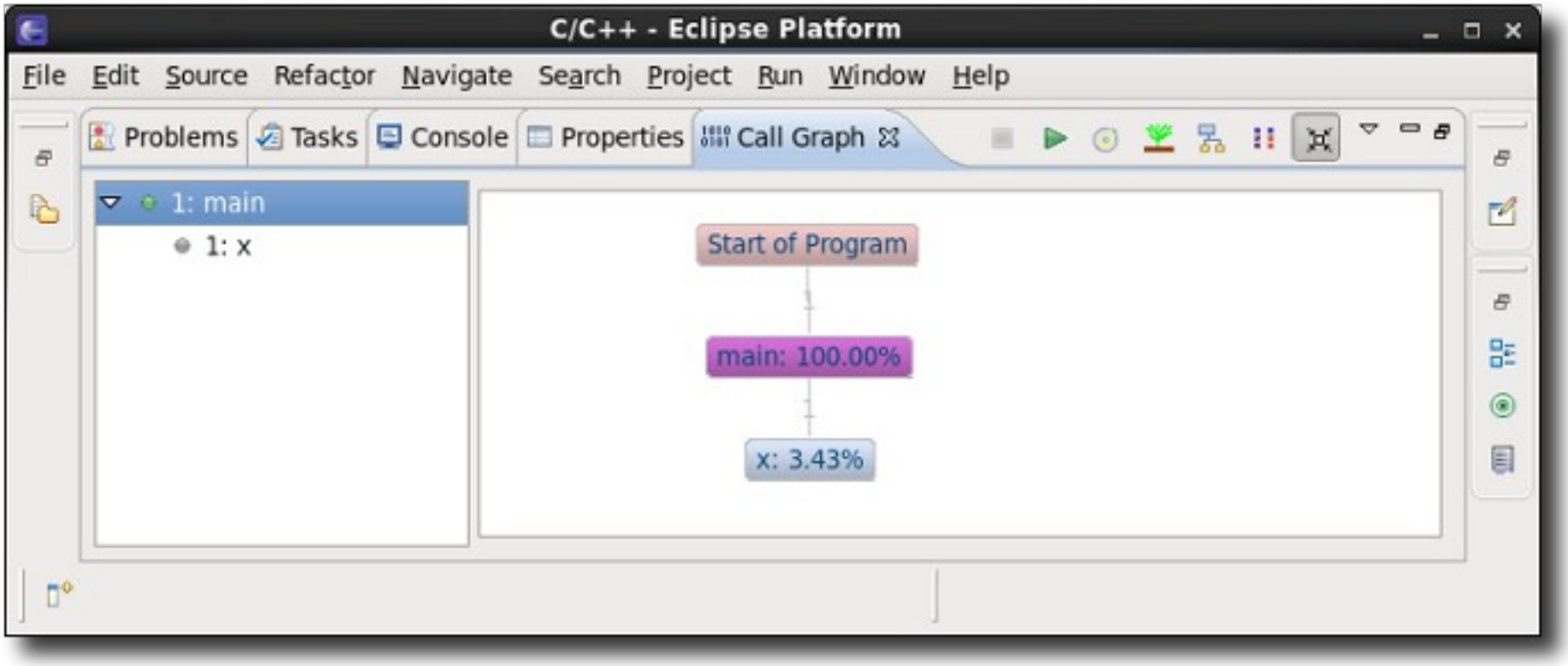
- Valgrind View shows charts and errors
- Can change type of tool used in Profile Configuration
  - Profile As -> Profile Configurations...
- Memcheck tool
  - Check for storage that isn't properly freed
  - Errors are marked in C/C++ editor as well as in Valgrind View
  - Can clear errors using X icon
- Massif tool
  - Shows chart of storage from start to finish of program
- Cachegrind tool
  - Charts cache misses and branch mispredictions for application

Oprofile

- Requires root access to run (opcontrol binary)
  - Root password will be asked for
- Set-up required before first profiling under Eclipse
  - Find install location of org.eclipse.linuxtools.oprofile.core plug-in
  - Go to natives/linux/scripts directory and run “sh ./install.sh” as root
- Oprofile options found in Profile Configuration
  - Default Profile As -> Oprofile performs CPU_CLK_UNHALTED
    - Snapshot at various intervals (sometimes no data collected)
- If data collected, shown in Oprofile View as tree
  - Opening tree reveals percentage times in code
  - Double-clicking on locations will bring up editor where location known

Callgraph (Function Call Tracking)

- Profile As -> Function callgraph
  - Requires SystemTap under the covers to run
  - Results shown in Call Graph View



Mylyn – Task Management

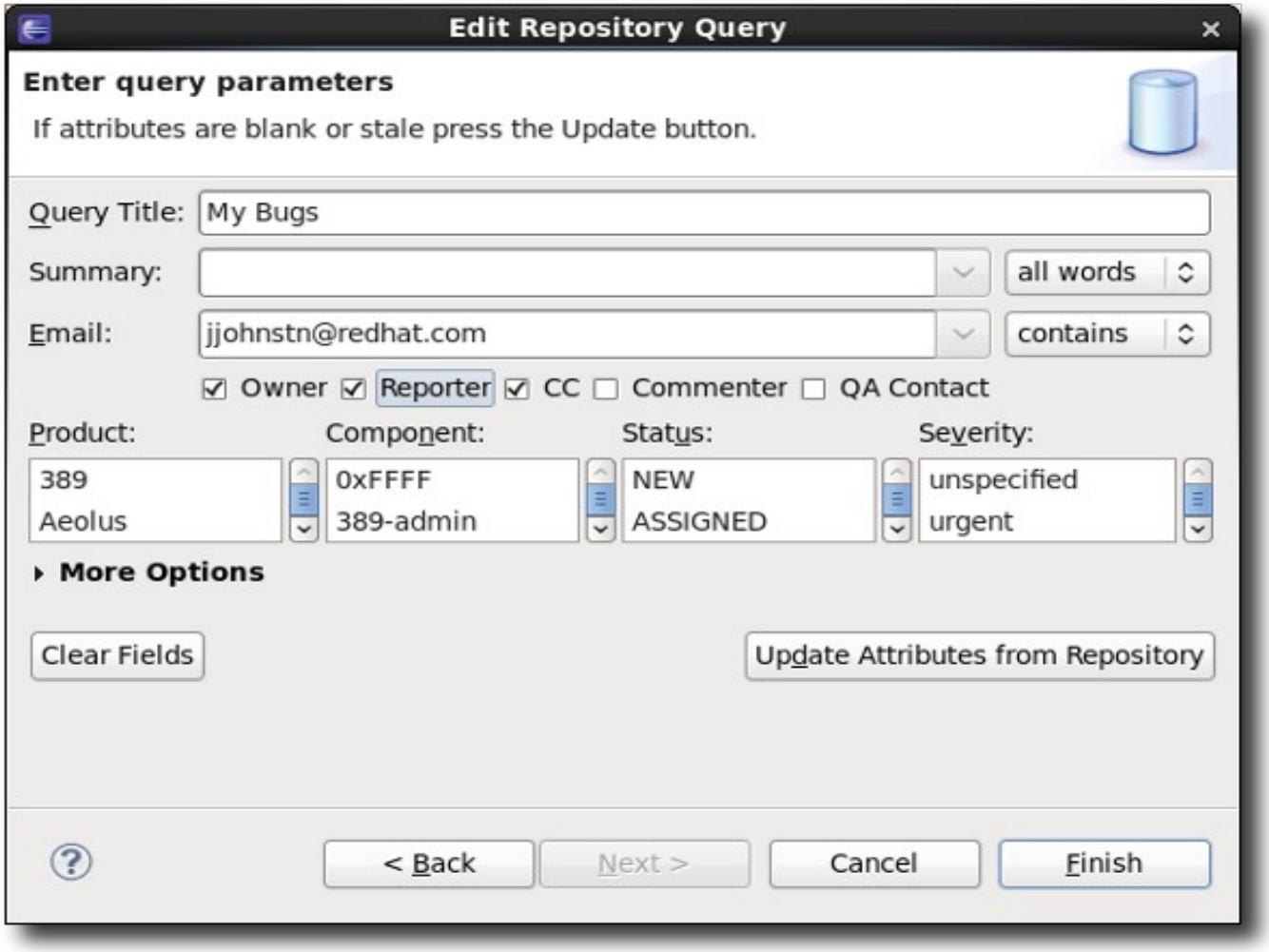
- Shipped in RHEL eclipse-mylyn* packages
- Task time
  - Can set deadlines for each task
  - Can track time taken on each active task
- Tasks have context
  - This includes all open files in editors and functions being referenced
  - Switching tasks restores the context of that task
- Can connect to task repositories
  - Can be Bugzilla (e.g. Eclipse bugzilla) or Trac
  - Can set up queries – each bug found is one task
  - Can perform changes off-line and synchronize later

Mylyn – Continued..

- Tasks are shown in Task List View
  - Window -> Show View -> Task List
  - Not to be confused with Tasks View which is for Todo items in files
    - e.g. //TODO: fix this line later
  - Can filter (scheduled, current work week, hide completed)
  - Can create a new task (through repository or local task)
- Task Repositories View shows repositories
  - Window -> Show View -> Other... -> Tasks -> Task Repositories
  - Can add new repositories or remove ones

Mylyn Continued..

- Task Query



RPM Spec Editor

- RHEL eclipse-rpm-editor package
- Default editor for .spec files
- Colourization and outline view
- Hover support for macros (e.g. %version)
- Completion support for package names
  - e.g. When adding Requires or BuildRequires statements
- Ctrl + Alt + C will add new changelog entry
- Can invoke rpmlint (Right-click in editor)
- Can create new spec file from template
  - New -> Other... -> RPM -> Specfile based on a template

RPM Spec Editor Continued..

- Can create rpm project
  - New -> Other... -> RPM -> RPM Project
  - Creates RPMS, SOURCES, SPECS, SRPMS folders
- Can export a binary/source rpm
  - Right-click -> Export... -> RPM -> Source/Binary RPM

Potential Future Enhancements

- Upgrade Eclipse component base (Indigo/Juno)
- Git support (egit/jgit upstream plug-ins)
- Perf profiling support
- Gcov/gprof support
- Rpm packaging (like eclipse-fedorapackager)